Experiment 2

Web Scrapping (Wikipedia)

Name:	**:**	**Abhishek N N**	**Register No**	**:**	**20BCE1025**
Faculty	:	Dr. Alok Chauhan	Slot	:	L51-L52 AB1-605B
Course	:	Web Mining Lab	Code	:	CSE3024
Programme	:	B.Tech CSE Core	Semester	:	Win – 22 - 23

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

https://en.wikipedia.org/wiki/Wikipedia

this link is used for lab experiment

Extraction of html code

This codes are taken from python notebooks so output will be shown next to it

```
import requests
r = requests.get('https://en.wikipedia.org/wiki/Wikipedia')
r.status_code
```

```
200
```

```
r.headers['content-type']
```

```
'text/html; charset=UTF-8'
```

```
r.encoding
```

```
'UTF-8'
```

```
# output is large so printing only first 2000 characters
r.content[:2000]
```

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en" dir="ltr">\n<head>\n<meta charset="UTF-8"/>\n<title>Wikipedia -
Wikipedia</title>\n<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":false,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":
["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":
["","January","February","March","April","May","June","July","August","September","October","November","December"],"wgRequestId":"9616912b-f218-4566-a57e-
11b6e1303f29","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":false,"wgNamespaceNumber":0,"wgPageName":"Wikipedia","wgTitle":"Wikipedia","wgCu
["*"],"wgCategories":["Pages using the Graph extension","Articles with hAudio microformats","Pages including recorded pronunciations","CS1 German-language sources
(de)","CS1 errors: missing periodical","CS1 maint: url-status","Pages containing links to subscription-only content","Webarchive template wayback links",\n"CS1 maint:
unfit URL","CS1 Italian-language sources (it)","CS1 Spanish-language sources (es)","All articles with dead external links","Articles with dead external links from
February 2022","Articles with permanently dead external links","Articles with short description","Short description matches Wikidata","Wikipedia indefinitely semi-
protected pages","Wikipedia indefinitely move-protected pages","Use mdy dates from December 2022","Use American English from June 2019","All Wikipedia articles
written in American English","Articles containing potentially dated statements from 2022","All articles containing potentially dated statements","Articles containing
potentially dated statements from November 2020","Articles containing potentially dated statements from January 2019","Articles containing potentially dated
statements from March 2020","Wikipedia articles ne'
```

Extraction of different components of the page

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(r.content, 'html.parser')

# helper function which returns all the html tags in a list
%run helpers_20BCE1025.ipynb
htmlTagsList = getHTMLTags()

for tags in htmlTagsList:
    print("-----------------------------------------------------")
    print(tags[0])  # printing tag category
    print("-----------------------------------------------------")
    for tag in tags[1:]:
        fl=soup.find_all(tag)
        if  len(fl)==0:
            continue
        print("------------")
        print(tag) # printing tag name
        print("------------")
        for f in fl:
            print(f.text.strip())  # printing tag content
        print("------------")

    print("-----------------------------------------------------")
```

```
Output exceeds the size limit. Open the full output data in a text editor
-----------------------------------------------------
Basic HTML
-----------------------------------------------------
------------
head
------------
Wikipedia - Wikipedia
------------
------------
title
------------
Wikipedia - Wikipedia
------------
------------
h1
------------
Wikipedia
------------
------------
h2
------------
Contents
History
Openness
Policies and laws
```

Error Handling

Error During Fetching of Website

When we are fetching any website content we need to aware of some of the errors that occur during fetching. These errors may be **HTTPError, URLError, AttributeError, or XMLParserError**. Now we will discuss each error one by one.

HTTPError:

HTTPError occurs when we're performing web scraping operations on a website that is not present or not available on the server. When we provide the wrong link during requesting to the server then and we execute the program is always shows an Error **"Page Not Found"** on the terminal.

Code:

```
import requests
from urllib.error import HTTPError

def webScraping(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
    except requests.exceptions.HTTPError as err:
        print("failed")
        print(err)
    else:
        print("success")


url1 = 'https://www.geeksforgeeks.org/implementing-web-scraping-
python-beautiful-soup/'
url2 = 'https://www.geeksforgeeks.org/page-that-do-not-exist'


webScraping(url1)
print("----------------------------------------------------------")
webScraping(url2)
```

Output:

```
success
----------------------------------------------------------
failed
404 Client Error: Not Found for url: https://www.geeksforgeeks.org/page-that-do-not-exist/
```

URLError:

When we request the wrong website from the server it means that URL which we are given for requesting is wrong then URLError will occur. URLError always responds as a server not found an error.

Code:

```
import requests
from urllib.error import URLError
link = 'https://www.amaz.in/s/ref=nb_sb_ss_ts-doa-
p_3_3?url=search-alias%3Daps&field-
keywords=basketball&sprefix=bas%2Caps%2C458&crid=3STPJQX67B7GD'
try:
  response = requests.get(link)
except URLError as url_error:
  print("Server Not Found")
else:
  print("There is no Error")
```

Output:

```
There is no Error
```

AttributeError:

The AttributeError in BeautifulSoup is raised when an invalid attribute reference is made, or when an attribute assignment fails. When during the execution of code we pass the wrong attribute to a function that attribute doesn't have a relation with that function then AttributeError occurs. When we try to access the Tag using BeautifulSoup from a website and that tag is not present on that website then BeautifulSoup always gives an AttributeError.

We take a good example to explain the concept of AttributeError with web scraping using BeautifulSoup:

Code:

```
import requests
import bs4
url = 'https://www.geeksforgeeks.org/implementing-web-scraping-
python-beautiful-soup/'

# getting response from server
response = requests.get(url)

# extracting html
soup = bs4.BeautifulSoup(response.text, 'html.parser')

# for printing attribute error
print(soup.NoneExistingTag.SomeTag)
```

Output:

```
AttributeError                            Traceback (most recent call last)
Cell In[22], line 13
     10 soup = bs4.BeautifulSoup(response.text, 'html.parser')
     12 # for printing attribute error
---> 13 print(soup.NoneExistingTag.SomeTag)

AttributeError: 'NoneType' object has no attribute 'SomeTag'
```

XML Parser Error :

We all are gone through XML parser error during coding the web scraping scripts, by the help of BeautifulSoup we parse the document into HTML very easily. If we stuck on the parser error then we easily overcome this error by using BeautifulSoup, and it is very easy to use.

When we're parsing the HTML content from the website we generally use **' xml '** or **' xml-xml '** in the parameter of BeautifulSoup constructor. It was written as the second parameter after the HTML document.

Code:

```
import requests
import bs4

url = 'https://www.geeksforgeeks.org/implementing-web-scraping-
python-beautiful-soup/'
response = requests.get(url)
soup = bs4.BeautifulSoup(response.text,'xml')

print(soup.find('div',class_='that not present in html content'))
```

Output:

```
None
```

HTML Parsing

Let's say you want to use Beautiful Soup look at a document's <a> tags. It's a waste of time and memory to parse the entire document and then go over it again looking for <a> tags. It would be much faster to ignore everything that wasn't an <a> tag in the first place. The SoupStrainer class allows you to choose which parts of an incoming document are parsed. You just create a SoupStrainer and pass it in to the BeautifulSoup constructor as the parse_only argument.
(Note that *this feature won't work if you're using the html5lib parser*. If you use html5lib, the whole document will be parsed, no matter what. This is because html5lib constantly rearranges the parse tree as it works, and if some part of the document didn't actually make it into the parse tree, it'll crash. To avoid confusion, in the examples below I'll be forcing Beautiful Soup to use Python's built-in parser.)

```
from bs4 import SoupStrainer

only_a_tags = SoupStrainer("a")
only_tags_with_id_link2 = SoupStrainer(id="link2")

def is_short_string(string):
    return string is not None and len(string) < 10

only_short_strings = SoupStrainer(string=is_short_string)

html_doc = """<html><head><title>The Dormouse's
story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters;
and their names were
<a href="http://example.com/elsie" class="sister"
id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister"
id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister"
id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
print(BeautifulSoup(html_doc, "html.parser",
parse_only=only_a_tags).prettify())
```

```
 <a class="sister" href="http://example.com/elsie" id="link1">
  Elsie
 </a>
 <a class="sister" href="http://example.com/lacie" id="link2">
  Lacie
 </a>
 <a class="sister" href="http://example.com/tillie" id="link3">
  Tillie
 </a>

print(BeautifulSoup(html_doc, "html.parser",
parse_only=only_tags_with_id_link2).prettify())

 <a class="sister" href="http://example.com/lacie" id="link2">
  Lacie
 </a>

print(BeautifulSoup(html_doc, "html.parser",
parse_only=only_short_strings).prettify())

 Elsie
 ,
 Lacie
 and
 Tillie
 ...
```